Parsing in a Multiprocessor Environment

Aron K. Insinga
Principal Software Engineer
Digital Equipment Corporation[1]
77 Reed Road, Hudson, MA 01749

Karen A. Lemone
Associate Professor of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Introduction

The compiling process by which a source program is translated to executable code is often divided into clearly discernible stages. The parsing stage finds the structure of a computer program as a precursor to the later stages leading to code generation. It is this stage that affirms much of the syntactic correctness of a program. In addition, parsing is a part of other language processors such as assemblers, syntax directed editors, etc.

In a uniprocessor, this procedure occurs in a straight-forward linear way token by token. When enough tokens have been amassed to identify the structure (e.g., as an expression, as an IF statement, etc.), we say that the structure has been parsed and *reduced* (to an expression, to an IF statement, etc.)

It has long been known that coprocessing of parsing, scanning (the stage that finds the tokens), and semantic analysis/synthesis (the rest of the translation process) decreases compiling time. With the current development of multiprocessors and distributed systems, it seems advantageous to explore how to best "run" all phases of a compiler in parallel. In addition, each phase, itself, may be effectively distributed among more than one processor.

This research focuses on parallel parsing using Parallel Piecewise-continuous LR parsing, a method proposed in [Mickunas 78] and [Schell 79]: the token stream is divided among n processors (it might be that these are the processors that found the tokens in the first place since parallel scanning is reasonable in a system employing parallel parsing.) Each parallel processor thus becomes a parallel parser and operates on a number of consecutive tokens.

Method

Given the BNF, a table generator can create an LR parsing table with the standard actions (*shift, reduce, accept, error*) plus two additional actions (*cancel, continue*) and additional states related to parallel processing. The extra actions involve shifting of terminal & nonterminal symbols to the parser which "can use them" (continuation) and reduction of partial structures (reduction and cancellation.) Since LR parsing is a left-to-right operation on the token stream, this means shifting to the "left" where the parallel parsers are ordered left to right according to the part of the token stream allocated to them. The reduction of partial structures occurs because a parser may be missing the left context for a structure. (We do not shift to the right.) A reduction (or a cancellation) with insufficient left context shifts a *cancellation symbol* to the "left." A cancellation symbol causes the reduction of the left part of the partial structure. A standard driver program with the added knowledge of what to do with these new actions operates on a segment of the input stream (of tokens and symbols transmitted from the "right") using the generated table.

We use Schell's method for both continuation and reduction of partial structures. Parsing continues until the token stream has been reduced to the "goal" symbol on the "left-most" parser.

[1] The views expressed here are those of the authors and not Digital Equipment Corporation.

Results

A parsing table for a simple arithmetic expression grammar was created based on the state machine in [Mickunas 78]. A parser driver was implemented and run with this table on an interpreter for a dialect of Pascal which supports parallel processes sharing memory.

We have discovered relationships between the token stream (`id+id+···id`) length, the number of processors, the size of the buffers used to pass messages between processors, and the parsing time and the number of messages transmitted. As might be expected, for "short enough" token streams and the overhead for a number of processors, more is not better. There is an optimal number of parsers for a given number of tokens; at this point, the speedup is about half of the number of processors. However, the number of messages transmitted is nearly linear with respect to the number of processors. Finally, there is a drastic decrease in the speedup when the size of the buffer is one message.

References

[Cohen 85] J. Cohen & S. Kolodner. "Estimating the Speedup in Parallel Parsing." *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 1, January 1985, pp 114-124.

[Insinga 86] Aron Klair Insinga. *Implementation of a Parallel Parser Driver*. M.S. thesis. Worcester Polytechnic Institute. 1986.

[Lipkie 79] Daniel Edward Lipkie. *A Compiler Design for Multiple Independent Processor Computers*. Ph.D. dissertation. University of Washington. 1979.

[Mickunas 78] M. Dennis Mickunas & Richard M. Schell. "Parallel Compilation in a Multiprocessor Environment (Extended Abstract)." *ACM: Proceedings of the Annual Conference*, 1978, pp 241-246.

[Schell 79] Richard M. Schell. *Methods for Constructing Parallel Compilers for use in a Multiprocessor Environment*. Ph.D. dissertation. University of Illinois, Urbana. 1979.